Executable Specifications

Foundations of Software Engineering
http://msrweb/foundations
April 16, 2002

The group

Partners

- BizTalk
- Hailstorm
- Indigo
- ITE (formerly TMT)
- Netconfig
- Palladium (formerly Trusted Windows)
- ...

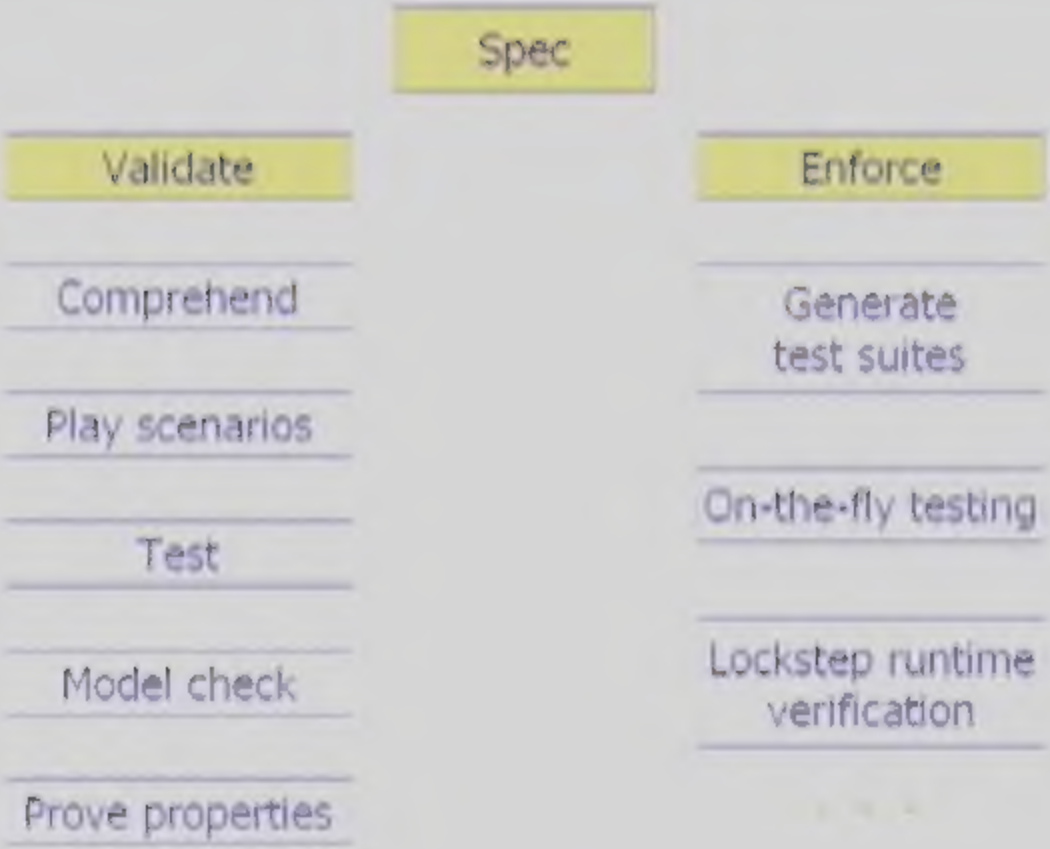
Problems we solve

- Poor specifications
  - Unenforceable
  - Poorly documented
- Growing spec. vs implementation gap
- High cost of design bugs
- Problems with testing
  - starts late in the process
  - often fails to test the intended functionality
  - is insufficiently automated

Our approach

Rigorous executable comprehensible spec

- Validate the design
  - Comprehend
  - Play scenarios
  - Test, model-check
  - Prove properties
- Enforce the spec
  - Generate test suites algorithmically
  - Use conformance execution


Spec
Validate
Enforce
Comprehend
Play scenarios
Test
Model check
Prove properties
Generate test suites
On-the-fly testing
Lockstep runtime verification

Conventional wisdom

- Specification should be declarative. Exec. spec is a contradiction in terms.
  - If you can execute the spec then why would you bother to implement it?
  - If you can execute the spec then it got to be full of irrelevant details, too specific, too low level.
- In fact, exec specs make sense

Example: topological sorting

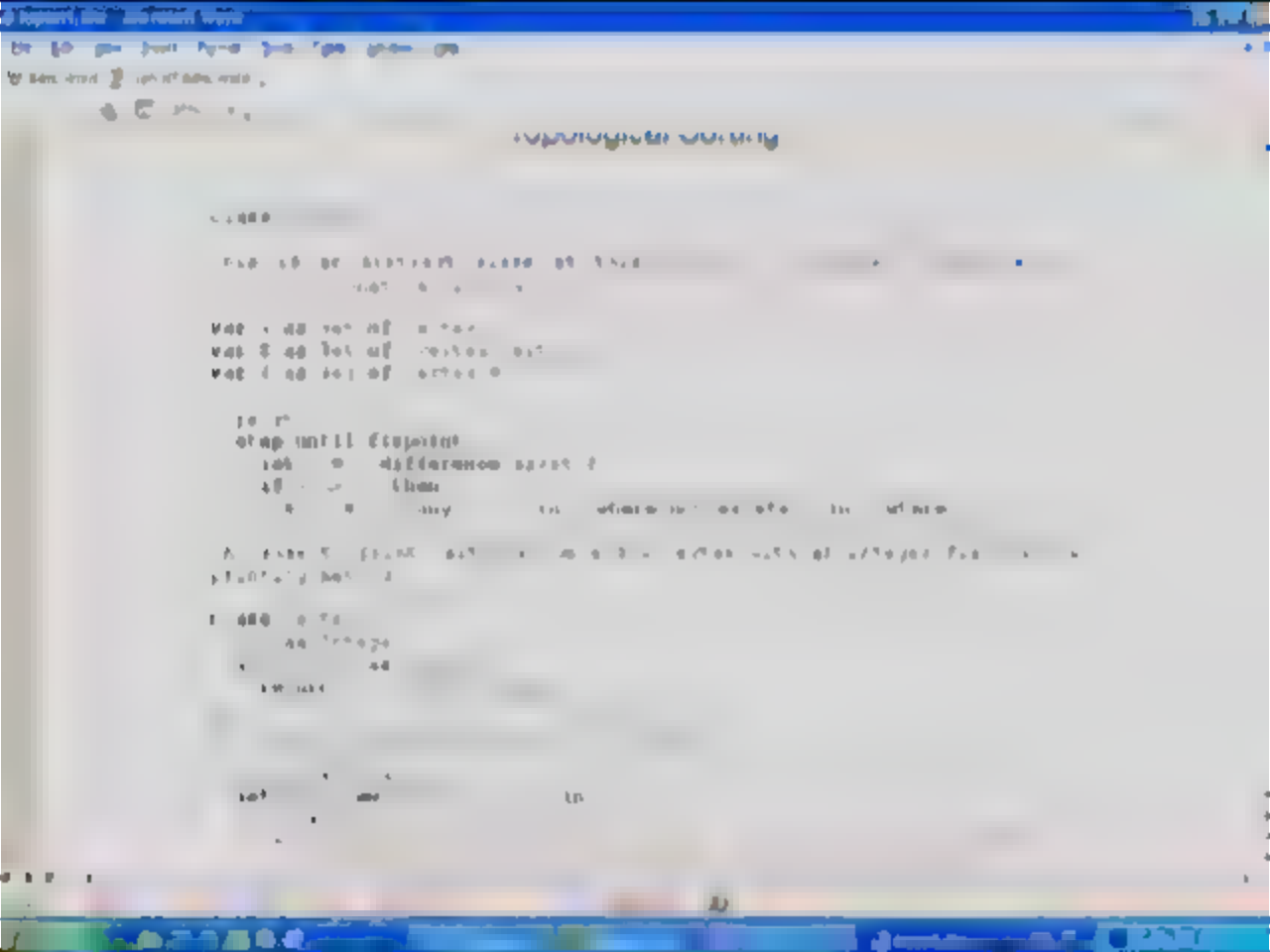
- Given an acyclic digraph G = (V,E), arrange the vertices into a sequence S where each edge (u,v) leads forward.
- Observe: there is a vertex without incoming edges, and the remaining digraph is acyclic.
- Topsort: Use the observation repeatedly to build the desired sequence S.

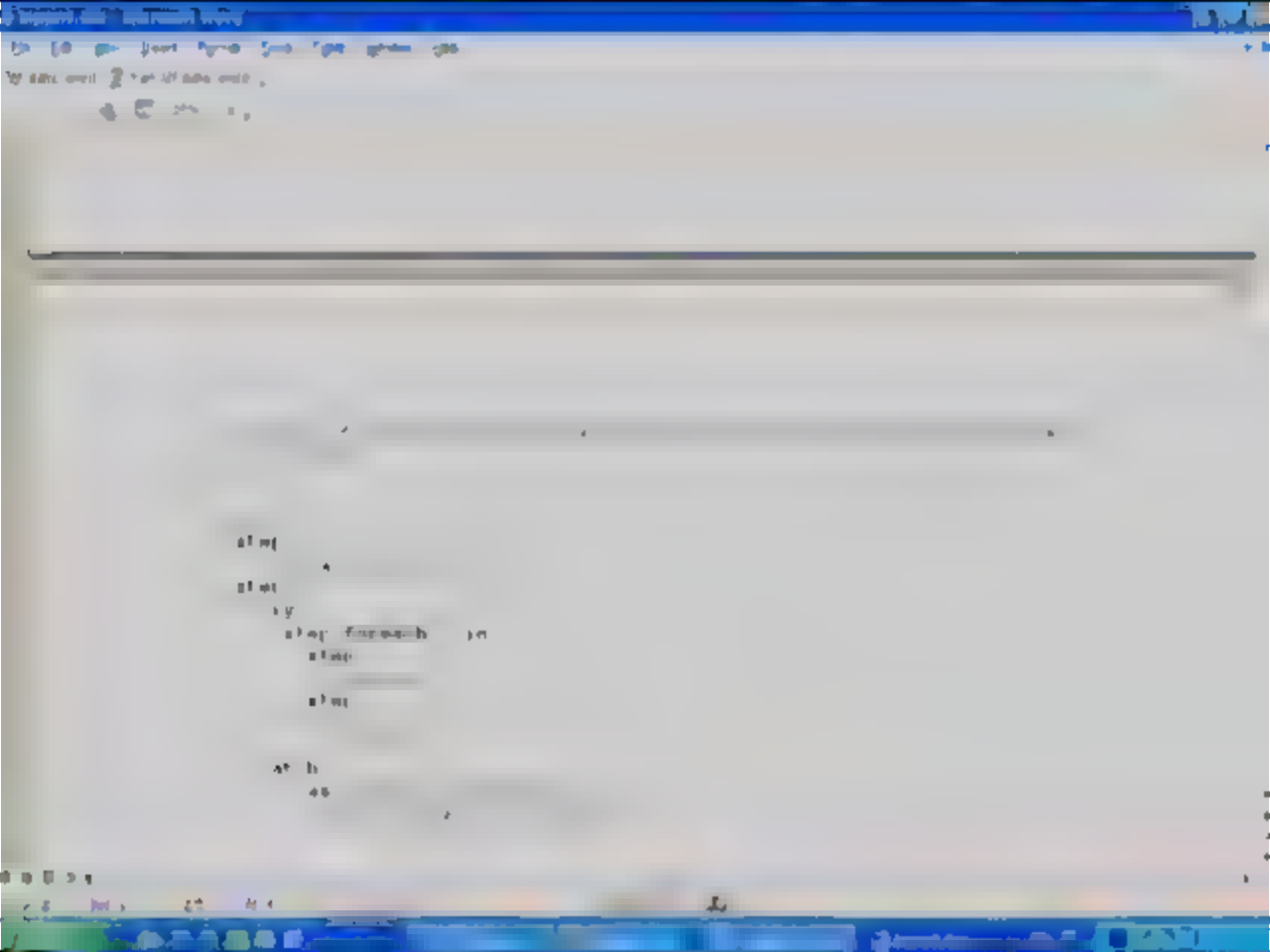
Topsort

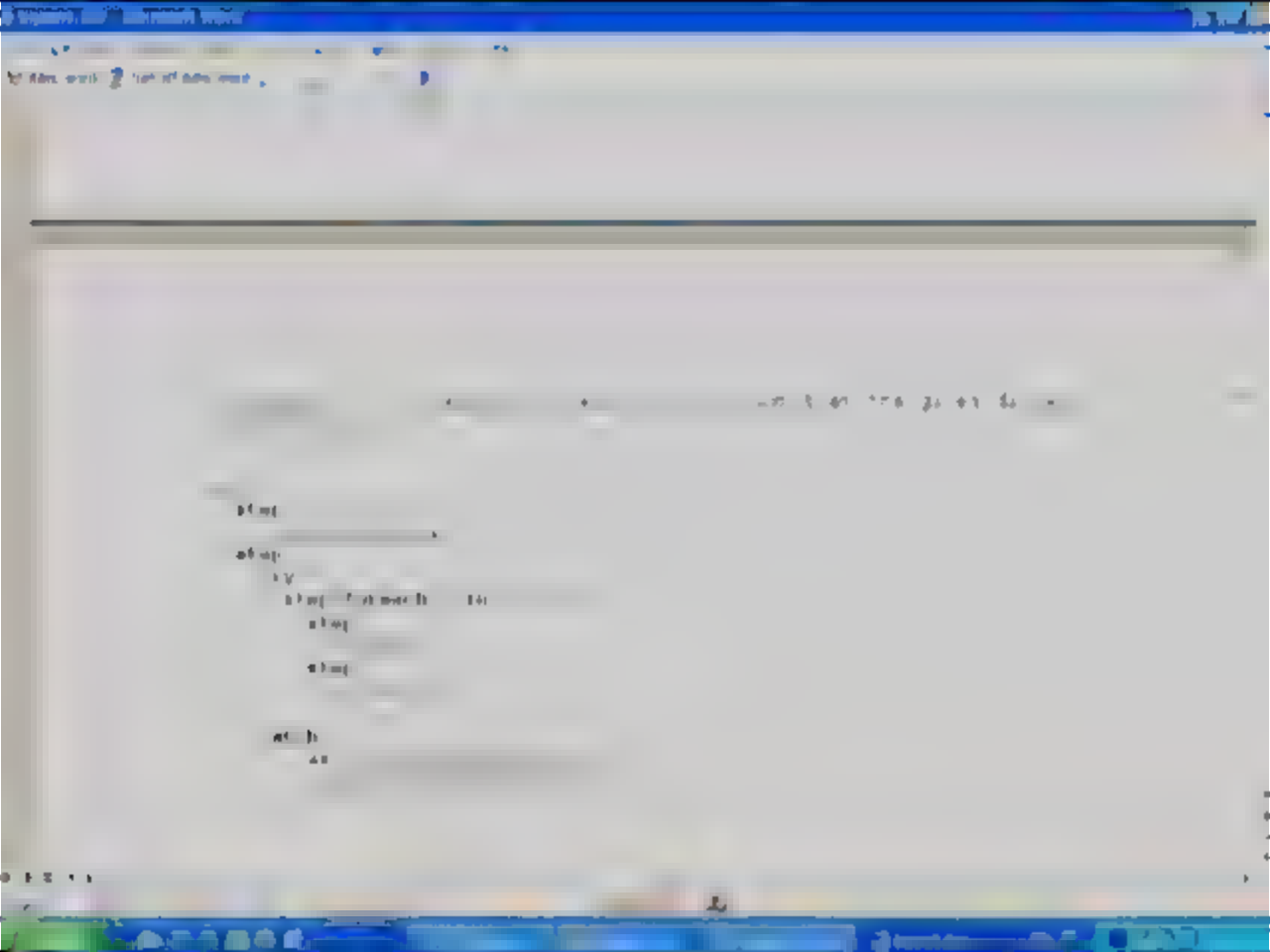
- A Modula-2 implementation by Niklaus Wirth
- An AsmL spec
  (a more OO version)

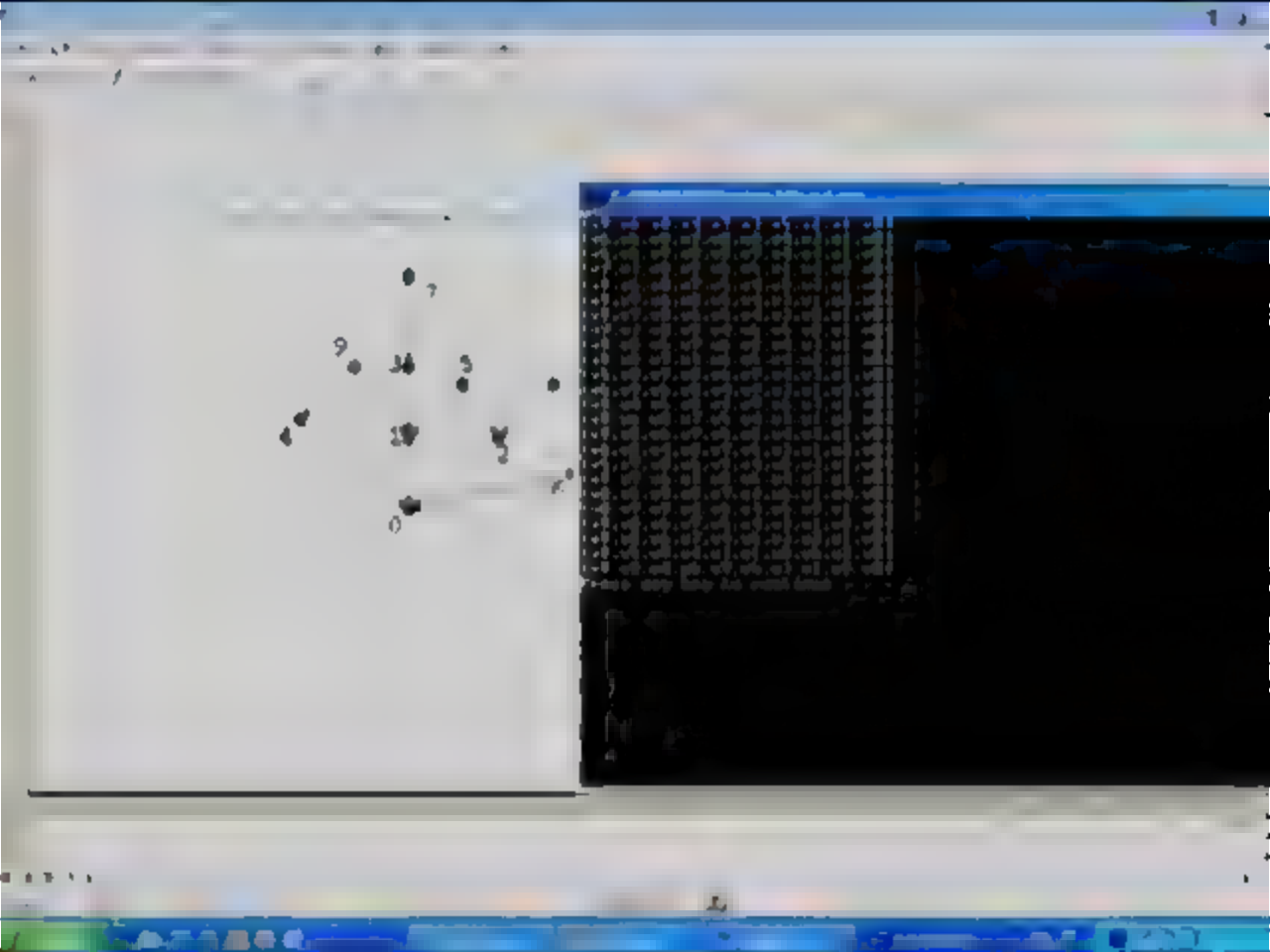
```
(* Read a sequence of relations defining a directed, finite
   graph.  Then establish whether or not a partial ordering
   is defined.  If so, print the element in a sequence showing
   the partial ordering. (Topological sorting)  *)

MODULE topsort;

FROM InOut   IMPORT WriteLn, WriteCard, ReadCard, WriteString;
FROM Storage IMPORT ALLOCATE;

TYPE lref = POINTER TO leader;

     tref = POINTER TO trailer;

     leader = RECORD
                key: CARDINAL;
                count: CARDINAL;
                trail: tref;
                next: lref;
              END;

     trailer = RECORD
                 id: lref;
                 next: tref
               END;

VAR head,tail,p,q: lref;
```

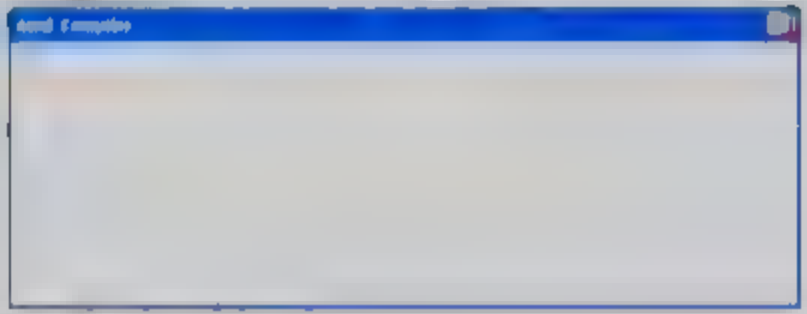
Topological Sorting

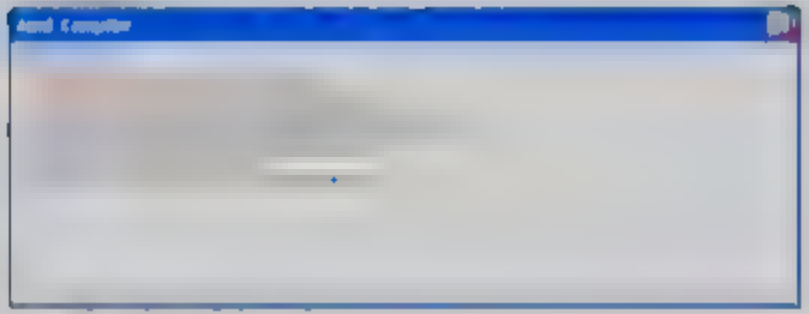


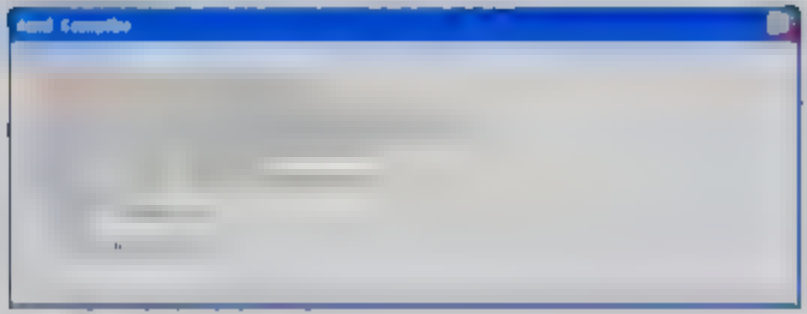


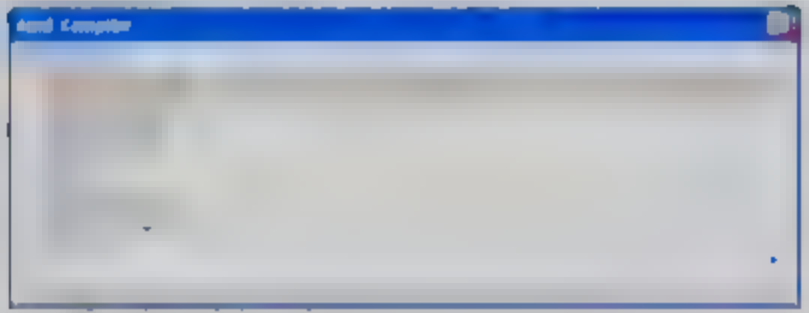


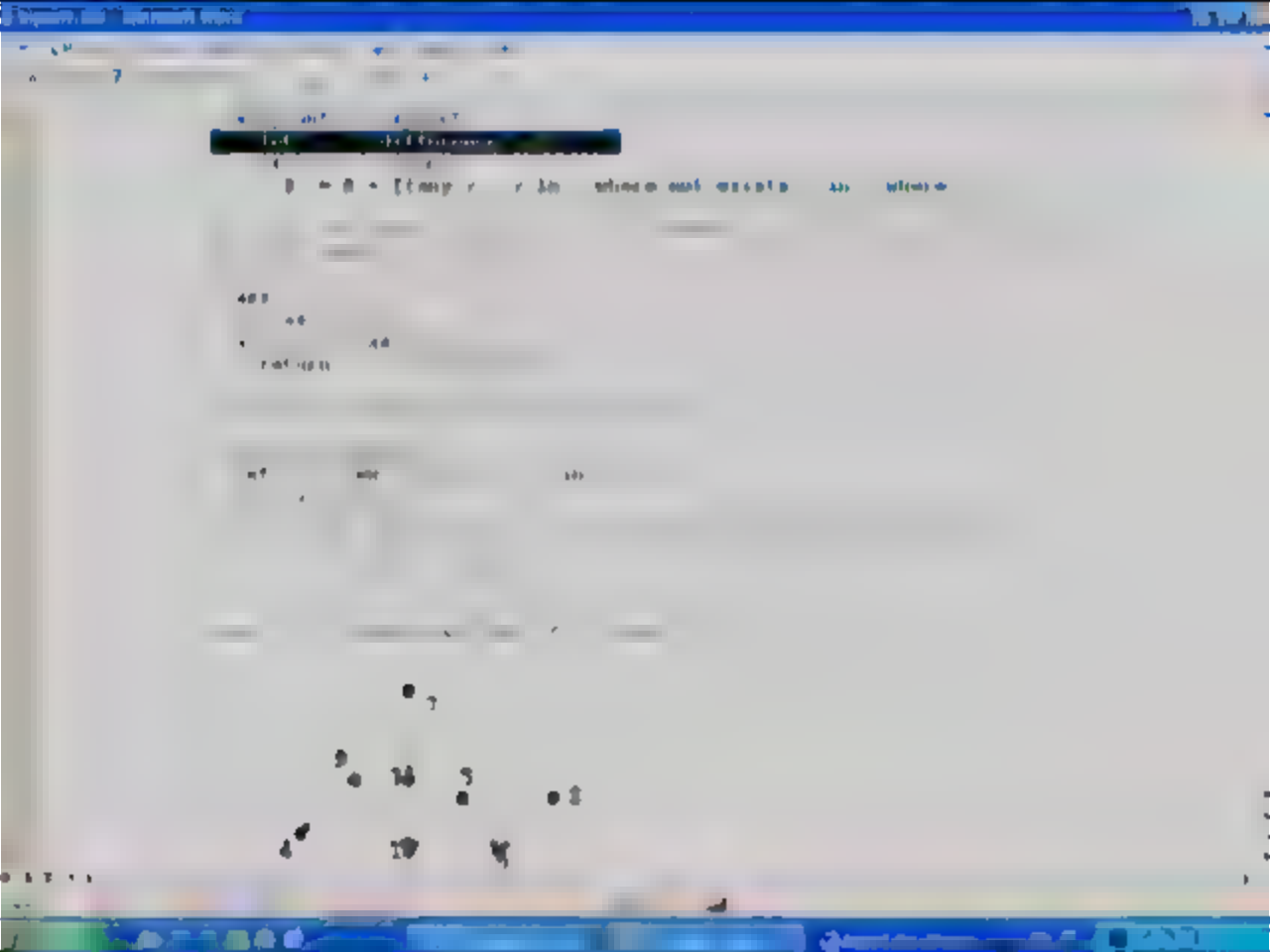


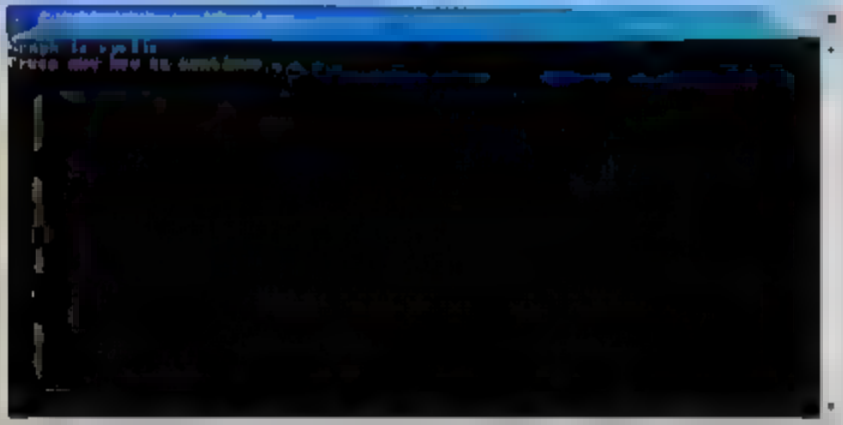






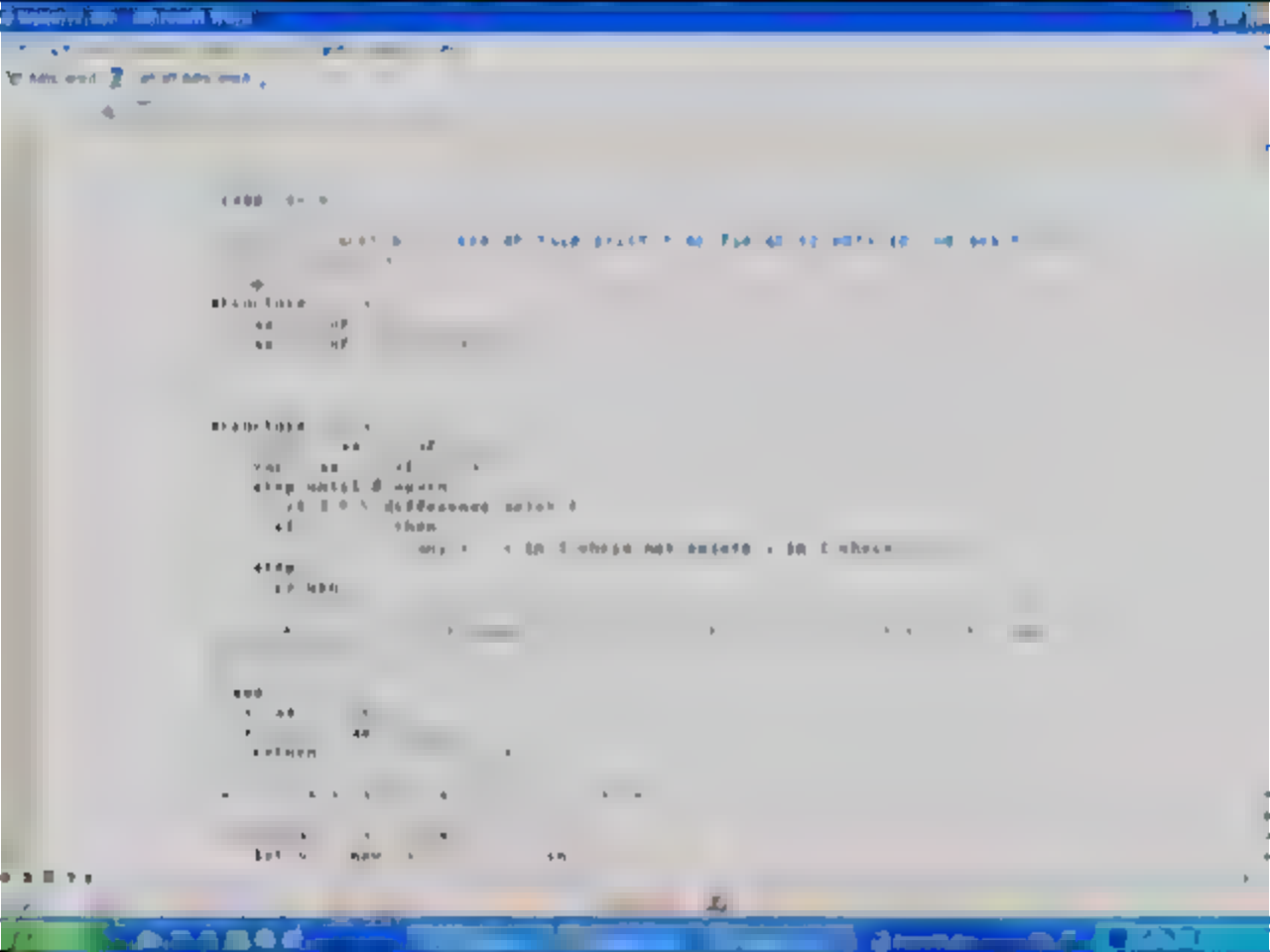






Topsort

-
-
( )



Theory and tools behind our approach

- ASMs: abstract state machines
- AsmL: ASM Language
- Additional tools for testing and runtime verification

ASMs

- Maximal expressivity of algorithms - including distributed algorithms - on their natural abstraction levels: For every algorithm A, there is an ASM B that is simulates A step for step.
  - Experimental confirmation
  - Seq and parallel non-distributed versions have been proved.

AsmL, Microsoft's ASM-Language

A powerful specification language

- [illegible]
- [illegible]
- [illegible]
- [illegible]

An example spec

- [illegible]

Page Table Edit Control
An AsmL Model

Foundations of Software Engineering
Microsoft Research

December 8, 2001

Abstract

1 Introduction

Foundations of Software Engineering

Microsoft Research

December 6 2001

Abstract

1 Introduction 1

1.1 Executive summary 1

1.2 Promises 2

1.3 Goals 3

Foundations of Software Engineering

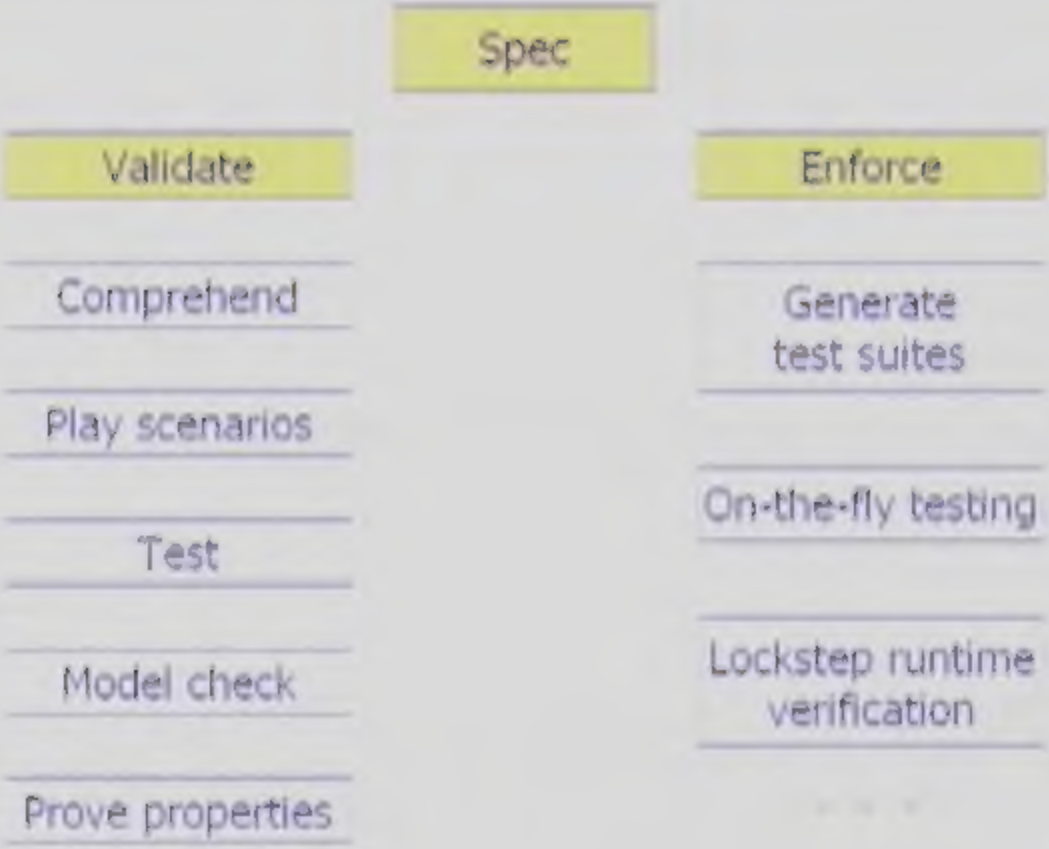
Microsoft Research

December 6 2001

Abstract

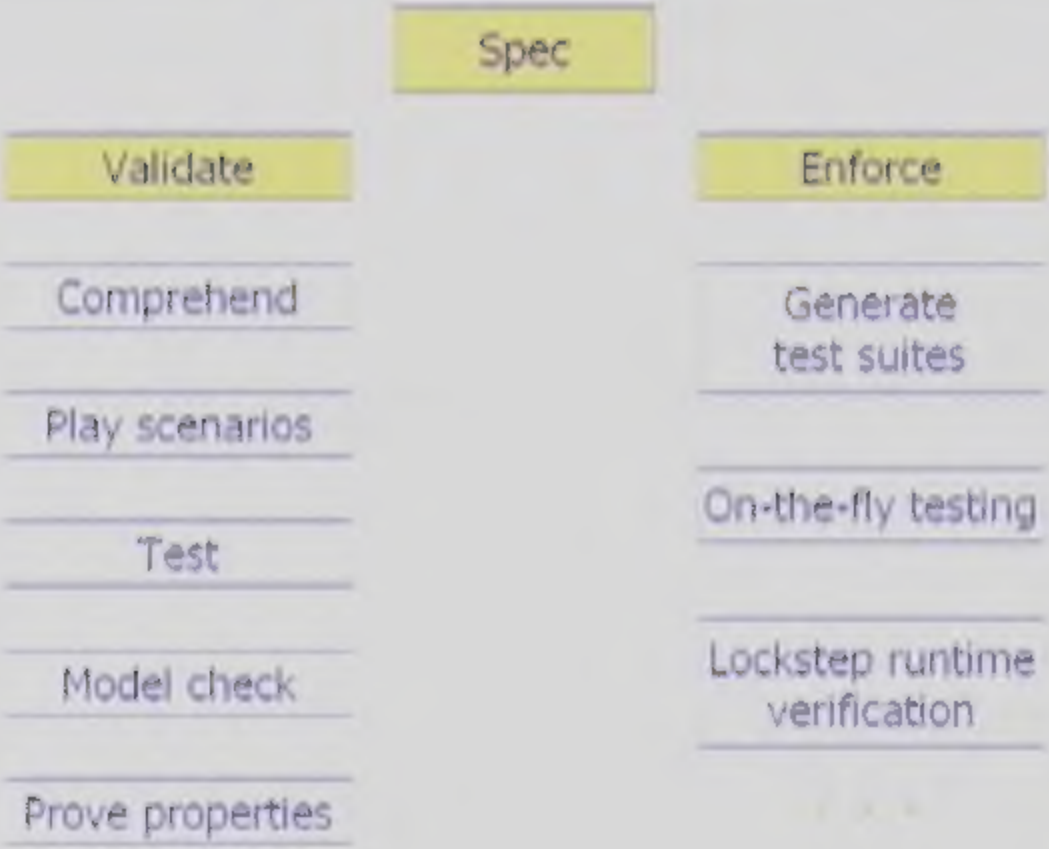
The asynchrony of buffered writes by multiple processors may be a security hole. An attack based on the asynchronous behavior of the system memory bus would proceed as follows:

- Figure out a way to manipulate the write buffer so that the writes of a particular processor drain into system memory as slowly as possible. For example, perform a large number of writes so that the processor's buffer is full.
- Select an unused page frame of system memory and format it as a page table. Choose some location *loc* in the new page as the target of attack. Write a word of memory into *loc* formatted as a page table entry whose target is a trusted page. Make sure that the operation goes into a "slow write buffer" from step one above.
- In parallel with the operation given in the previous bullet, add the page frame number of the page table from the previous step as a page table entry of some directory found in `System.RPV`. Do this in such a way that the page table edit will be resolved before the buffered write from the previous step has had a chance to be drained into system memory. This will require that two processors be used, one with a busy write buffer and one with a relatively idle one. (The attack cannot be performed with just one


Spec
Validate
Enforce
Comprehend
Play scenarios
Test
Model check
Prove properties
Generate test suites
On-the-fly testing
Lockstep runtime verification

Testing with AsmL

- We are deeply involved with testing.
  - Our approach empowers testers, and they like that.
- The ITE/AsmL project
- There are powerful testing techniques for finite automata
- An ASM-to-FSM algorithm


Spec
Validate
Comprehend
Play scenarios
Test
Model check
Prove properties
Enforce
Generate test suites
On-the-fly testing
Lockstep runtime verification